Armen Arakelian

NFT Wrapper

reaudit report

Content

Summary 2

Structure 2

Analysis 4

Vault721ERM_V01.sol 4

RESOLVED - the value is decreased now. 4

RESOLVED - the method was simplified. 4

RESOLVED - the method was removed. 5

RESOLVED - clarified in the document that nullification is required to protect against reentrancy and subsequent method execution. 5

RESOLVED - the check has been rewritten and now uses underflow to save the gas. 6

RESOLVED - the methods have been rewritten to save the gas. 7

Vault1155ERM_V01.sol 9

RESOLVED - the method was simplified. 9

RESOLVED - the recommended requirements have been added to appropriately handle the overflow cases. 9

RESOLVED - the method was removed. 9

RESOLVED - the check has been rewritten and now uses underflow to save the gas. 10

RESOLVED - the methods have been rewritten to save the gas. 10

Summary

During the reaudit, all the suggested changes were applied and no other bugs have been found. The contracts are in good shape to be deployed.

Structure

- **Informational** - The issue has no impact on the contract's ability to operate.
- **Low** - The issue has minimal impact on the contract's ability to operate.
- **Medium** - The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.
- **High** - The issue affects the ability of the contract to compile or operate in a significant way.
- **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

Analysis

Vault721ERM_V01.sol

RESOLVED - the value is decreased now.

High: The value is increased instead of decreasing.

```
function _withdrawToken(address destination, uint256 index) internal {
    . . .

    // Decrease the tracked count for this contract.
    _contractTokenCounts[contractIndex - 1]++;

    . . .
}
```

Recommendation: Decrease the value instead.

RESOLVED - the method was simplified.

High: An admin can accidentally set up an undesired role for a user.

```
function setRoles(address[] calldata users, bool enableRoles, bool
    adminFlag, bool depositorFlag, bool withdrawerFlag) external {
    . . .

    uint256 rolesFlag;
    if (depositorFlag) {
        rolesFlag |= R_CAN_DEPOSIT;
    }
    if (withdrawerFlag) {
        rolesFlag |= R_CAN_WITHDRAW;
    }
    if (adminFlag) {
        rolesFlag |= R_IS_ADMIN;
    }

    address user;
    uint256 count = users.length;
    uint256 i;
```

```
        for (; i < count; i ++) {
              user = users[i];
              if (enableRoles) {
                    _addressRoles[user] |= rolesFlag;
              } else {
                    _addressRoles[user] &= ~rolesFlag;
              }
        }

        . . .
}
```

For instance, an admin wants to **disable** *depositor* and *withdrawer* roles for a set of users. To achieve that, an admin would call this function like so:

```
setRoles(someUsers, false, false, true, true)
```

But then, he would unintentionally set up an *admin* role for the given set of users.

`adminFlag` is `false`, so `R_IS_ADMIN` flag would not be set inside `rolesFlag`. In the loop, because `enableRoles` is `false`, `_addressRoles[user]` would receive negated `rolesFlag`, but because we did not set `adminFlag` there, a user would become an admin.

Recommendation: rewrite a function to make it more transparent.

RESOLVED - the method was removed.

Medium: The `changeContractAddress` function doesn't check if `newContract` already added to the vault.

Recommendation: apply `_coreAddressIndices[newContract] == 0` check

RESOLVED - clarified in the document that nullification is required to protect against reentrancy and subsequent method execution.

Low: Some gas could be saved if _depositor kept preserved.

```
function deposit(address depositor, uint256[] calldata tokenIds,
address[] calldata tokenContracts) external {
      . . .
```

```
    _depositor = msg.sender;

    . . .

    _depositor = address(0);

    . . .
}
```

The sstore operation costs 20k gas to change the value from zero to non-zero and 5k gas to keep the value unchanged or zero. 15k gas refund is given for changing a value from non-zero to zero. That means one will pay 20k + 5k - 15k = 10k gas with the current implementation. If the contract omitted nullification, then one would pay 20k gas the first time, but only 5k gas subsequently, making the overall price cheaper (see page 25 here).

Recommendation: delete the nullification.

RESOLVED - the check has been rewritten and now uses underflow to save the gas.

Low: Double storage read (1).

```
function canDepositToken(address tokenContract, uint256 tokenId) public
view returns (bool) {
    // Get the filtering data for this contract. If there is no
    filtering data, then there is no restriction on filtering.
    uint256[] storage filteringDataStart =
    _filteringDataStart[tokenContract] ;

    if (filteringDataStart.length == 0) {
        return true;
    }

    // Do the binary search.
    uint256 low;
    uint256 high = filteringDataStart.length - 1;

    . . .
}
```

Recommendation: save the storage variable to a memory one.

RESOLVED - the methods have been rewritten to save the gas.

Low: Double storage read (2).

```
function withdrawTokenAtIndex(address destination, uint256 index)
external {
        require(
                _status == S_NOT_ENTERED
                && (
                        ((_roleRestrictions & WL_RESTRICT_WITHDRAWALS) !=
                        WL_RESTRICT_WITHDRAWALS)
                        || ((_addressRoles[msg.sender] & R_CAN_WITHDRAW)  ==
                        R_CAN_WITHDRAW)
                )
                && destination != address(0)
                && destination != address(this)
                && _coreAddressIndices[destination] == 0

                && _tokenIds.length > 0
                , "Invalid parameters");
        _status = S_ENTERED;

        _withdrawToken(destination, index);

        . . .
}


function withdrawTokens(address destination, uint256[] calldata
tokenIds, address[] calldata tokenContracts) external {
        . . .

        for (uint256 i; i < count; i ++) {
                tokenContract = tokenContracts[i];
                tokenId = tokenIds[i];
                uint256 contractIndex = _coreAddressIndices[tokenContract];
                uint256 tokenIndex = _indices[(contractIndex <<
                CONTRACT_INDEX_OFFSET) | tokenId];
                if (
                        contractIndex == 0 // Contract is not allowed in this
                        vault.
                        || tokenIndex == 0 // Token is not in this vault.
                        || _tokenContracts[--tokenIndex] != tokenContract
                        || _tokenIds[tokenIndex] != tokenIds[i]
                ) {
                        continue;
                }
                _withdrawToken(destination, tokenIndex);
                withdrawnCount++;
        }

        . . .
}


function _withdrawToken(address destination, uint256 index) internal {
        IERC721 tokenContract = IERC721(_tokenContracts[index]);
        uint256 tokenId = _tokenIds[index];
        uint256 contractIndex =
        _coreAddressIndices[address(tokenContract)];

        . . .
}
```

Recommendation: Rewrite functions to optimize gas cost.

Vault1155ERM_V01.sol

RESOLVED - the method was simplified.

High: An admin can accidentally set up an undesired role for a user.

---=== Same as in Vault721ERM_V01.sol. ===----

RESOLVED - the recommended requirements have been added to appropriately handle the overflow cases.

High: High possibility of overflow.

```
function deposit(address depositor, address[] calldata tokenContracts,
uint256[] calldata tokenIds, uint256[] calldata tokenCounts) external {
      . . .

      _contractTokenCounts[contractIndex - 1] += tokenCount;

      . . .

      mintCount += tokenCount;

      . . .

      if (mintCount > 0) {
            _mint(depositor, mintCount * _parityDepositAmount);
      }

      . . .
}
```

The `deposit` function is super unsafe to execute since the `tokenCounts` is unbounded and `SafeMath` is not used whatsoever. The snipped highlights code with the most dangerous places.

Recommendation: use `SafeMath` **at least** in the `deposit` function.

RESOLVED - the method was removed.

Medium: The `changeContractAddress` function doesn't check if `newContract` already added to the vault.

---=== Same as in Vault721ERM_V01.sol. ===----

RESOLVED - the check has been rewritten and now uses underflow to save the gas.

Low: Double storage read (1).

---=== Same as in Vault721ERM_V01.sol. ===----

RESOLVED - the methods have been rewritten to save the gas.

Low: Double storage read (2).

---=== Same as in Vault721ERM_V01.sol. ===----